Proceedings of the 3rd International Conference on Software Engineering and Machine Learning
DOI: 10.54254/2755-2721/2025.30674

Analyzing Approaches to Extending and Implementing Behavior Trees in Computer Games

Siyun Guo[1], **Sitong Hou**[2*]

[1] *Inner Mongolia University of Technology, Hohhot, China*
[2] *Beijing University of Posts and Telecommunications, Beijing, China*
Corresponding Author. Email: hst339@bupt.edu.cn

Abstract. With the development of the computer game industry, it is too hard for traditional methods, like finite state machines and behavior trees, to drive intelligent non-player characters behaviors, especially under complex environments. This paper explores and analyzes three methods - an event driven approach to behavior trees, simulating human-like emotions in behavior trees, and integrating machine learning in behaviour trees and allowing them to evolve themselves through genetic programming. This review synthesizes the findings from existing studies, highlighting the potential of these extensions to significantly enhance player immersion and the authenticity of gaming experiences.

Keywords: behavior trees, game algorithm, video games

1. Introduction

As technologies advance, computer game developers strive to enhance the responsiveness of behavior in non-player characters (NPCs). Historical approaches - finite state machines and behavior trees, which lacked variability and only had the ability to execute action based on simple true-or-false conditions - are no longer sufficient to fulfill the standards of consumers looking for a human-like interaction with NPCs.

New and innovative extensions and improvements proposed to behavior trees may push the computer game industry into a new era.

This paper analyzes three innovative extensions to behavior trees aimed at improving the interactivity and realism of NPCs. Firstly, Event-Driven Behavior Trees (EDBTs) introduce coordination nodes that facilitate sophisticated interactions among NPCs, addressing scalability issues while maintaining the intuitiveness and reusability of behavior trees. Secondly, Emotional Behavior Trees integrate simulated human emotions into the decision-making process, adding a new dimension to the variety and human-like behavior of NPCs. Lastly, the evolution of behavior trees through genetic programming simulates natural selection processes, generating complex and human-like behaviors that adapt to different gaming environments and player strategies.

2. Background

Behavior Trees (BTs) represent a significant advancement in the field of artificial intelligence, particularly within the domain of computer game development, where they serve as a sophisticated mecha-

DOI: 10.54254/2755-2721/2025.30674

nism for directing the actions of NPCs. While the traditional methods, such as Finite State Machines (FSMs), are straightforward to implement, they often falter when faced with the intricate demands of contemporary game design, where NPCs are expected to exhibit a rich variety of dynamic behaviors. In contrast, BTs offer a hierarchical and scalable approach that transcends these limitations, enabling developers to craft complex decision-making processes with relative ease.

At the core of a BT is its structured hierarchy, commencing with a root node that diverges into a network of child nodes representing different behaviors or conditions. These nodes are classified into decision-making units like selectors, which determine the execution path by evaluating child nodes, and sequential nodes, which execute their children in a predefined order. The leaves of the tree are typically action or condition nodes, representing the atomic behaviors that NPCs can perform or check against the game's state.

The execution model of a BT is both elegant and efficient, employing a depth-first search that navigates through the tree from the root to the leaves, executing nodes along the way. This process is governed by the outcomes of each node—success, failure, or running—which collectively influence the flow of the tree's decision-making cascade. BTs offer scalability, unlike FSMs that struggle with growing complexity. They use composite nodes for complex behaviors and remain stateless, simplifying developmental growth. BTs are also highly expressive, mimicking intricate human decision-making with layered control structures and advanced nodes, enriching NPC behavior. Their flexibility allows for simultaneous behavior execution and action preemption based on priority or conditions, crucial for NPCs handling multiple tasks or adapting to changes swiftly. In a game environment, the execution of a BT is typically integrated into the game's update loop, which runs repeatedly every frame or at regular intervals. The specifics of how a BT is executed can vary depending on the game's design and the requirements of the NPC.

This article will cover the work on extending behavior trees in the aspects of NPC collaboration, the influence of emotional factors, and the method of generating behavior trees using genetic algorithms.

3. An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games

[1] introduces an extension to Event-driven Behavior Trees (EDBTs). The authors propose three new types of nodes: Soft Request Sender, Hard Request Sender, and Request Handler, which allow NPCs to send and receive requests. They also present a methodology for adequately structuring each agent's EDBT using the coordination nodes while following some desirable principles to facilitate multi-agent coordination without deviating from the behavior tree development paradigm.

3.1. Event-driven behavior trees

EDBTs evolved to address scalability issues and complex behavior demands in commercial video games. They enhance traditional BTs by introducing nodes capable of responding to events and aborting running nodes. EDBTs facilitate the design of larger, deeper trees without performance penalties through a scheduler that stores and updates previously active behaviors, avoiding the need to traverse the tree from the root every frame. EDBT is also a directed tree where leaves represent possible actions and inner nodes represent decision-making processes. when an event occurs, it triggers the execution of a specific branch, proceeding that branch by traversing downward based on the unique execution rules of each type of node, or it may terminate based on the outcome of operations. There

DOI: 10.54254/2755-2721/2025.30674

are some key components of EDBTs include: Composite Nodes, Service Nodes and Blackboards, and Decorator Nodes. An example of which is as shown in Fig. 1.

Composite nodes include selector nodes that sequentially tick(traverse and execute the nodes in a predefined sequence) children until success, sequencer nodes that continue until failure, and parallel nodes that execute children simultaneously, allowing concurrent subtree executions.

Service nodes tick a child node and repeatedly call a method, often used for checks and updates to the blackboard, a shared data structure that stores key-value pairs for referencing by nodes.

Decorator nodes in EDBTs enhance child nodes by adding behavior or conditions without altering the nodes themselves. They include conditional decorators that execute child nodes based on met conditions, conditional loop decorators that repeat executions if conditions persist, and loop decorators that re-trigger nodes a set number of times or continuously. And the blackboard observer decorators(BOD), a specialized type that emerged with EDBTs, these nodes are use to monitor events and have the capacity to halt active nodes, responding to changes quickly.

These key elements work in concert to enable traditional BTs to respond to events and terminate running nodes, transforming them into EDBTs. But, EDBTs (and BTs) focus on the creation of individual behaviors. Therefore, the authors introduce three new types of nodes. In the following subsection, the distinct responsibilities and operational mechanisms of these nodes will be explained, and it will be clarified what types of nodes they originally belong to.

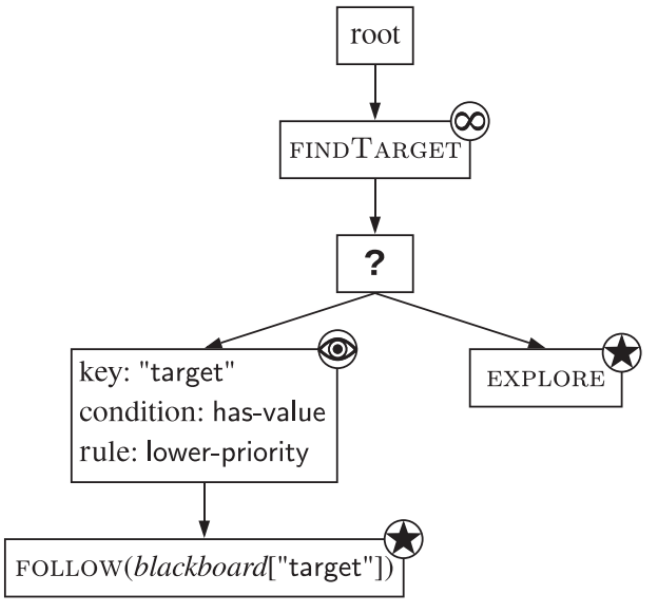


Figure 1: Event-driven behavior tree [1]. Nodes are represented by rectangles. Nodes' type is depicted with a symbol either inside the rectangle or in the top-right corner. Service nodes are represented by an infinity symbol, selector nodes by a question mark, BOD nodes by an eye, and task nodes by a star.

3.2. Coordination Nodes and Methodology

The authors introduce the three coordination nodes which are an extension to EDBTs. This extension aims to improve the complexity and realism of NPC behaviors, providing a more engaging experience for players by allowing NPCs to coordinate effectively without the need for complex individual

DOI: 10.54254/2755-2721/2025.30674

behaviors or hard-coded coordination logic. These nodes allow NPCs to communicate and engage in collaborative actions through a messaging system involving requests.

The three types of nodes are as follow: Soft Request Sender (SRS) nodes, allowing an NPC to issue a soft request to other NPCs, which can be executed concurrently with the sender's own actions without requiring confirmation. This class of node is a task node. Hard Request Sender (HRS) Nodes are used for hard requests where the sender NPC awaits confirmation from a specified number of receivers before proceeding as a decorator node. This ensures synchronized action among a group. Request Handler (RH) Nodes are responsible for handling incoming requests within an NPC's behavior tree, executing the appropriate subtree based on the request. This class of nodes is a specialization of the BOD.

The coordination nodes maintain the benefits of EDBTs, such as visual intuitiveness, scalability, and reusability, while enabling more sophisticated and dynamic interactions between NPCs. The authors also present a methodology for integrating these nodes into an EDBT, emphasizing a clear separation between individual and coordinated behaviors and ensuring a logical priority order.

The methodology makes the resulting EDBTs visually intuitive while enabling NPCs to engage in coordinated behaviors effectively. There are four principles about the methodology:

Separation of Concerns: NPCs distinguish between individual and coordinated behaviors within the EDBT to maintain clarity and organization.

Priority Ordering: A very clear visual hierarchy of priorities should be established among the different types of requests that NPCs can handle.

Interruptibility: Non-critical nodes in the individual behaviors of NPCs that are in a running state can be terminated by any request, ensuring NPCs can respond flexibly to new situations.

Non-critical Subtree Protection: The non-critical nodes in the subtree under the RH node that are in a running state can only be terminated by requests of high priority.

The methodology also discusses how to manually control the `checkMailbox` process, which is responsible for handling incoming requests, and suggests structuring the EDBT to reflect these principles. This includes placing a service node that calls `checkMailbox` at a strategic location within the tree and organizing RH nodes in a manner that respects the priority order of requests.

By following this methodology, developers can create NPCs that exhibit both independent and coordinated behaviors in a way that is modular, maintainable, and aligned with the development paradigm of behavior trees. The result is a more dynamic and interactive gaming experience where NPCs can respond intelligently to the game's evolving context.

3.3. Example

In [1], coordination nodes are applied in a scenario with firefighter NPCs tasked to extinguish player-caused fires in a game. Effective fire extinguishment requires the coordinated effort of at least three firefighters. Individually, firefighters explore the terrain, but upon discovering a fire, one initiates a request for two peers to assist, ensuring the fire is extinguished collectively.

DOI: 10.54254/2755-2721/2025.30674

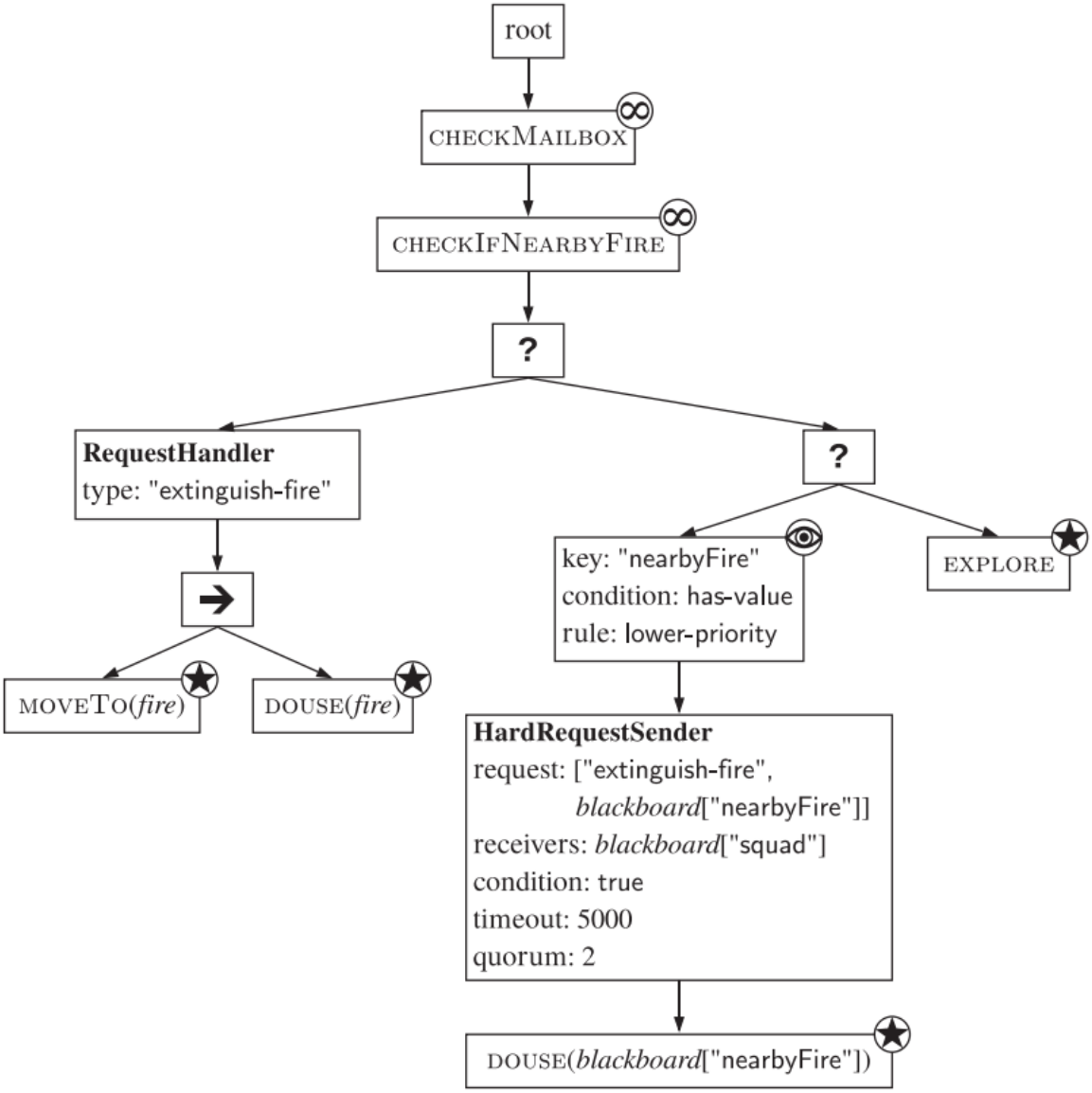


Figure 2: EDBT for the firefighter NPCs from Example [1].

As shown in Fig. 2, the process starts at the root, coordination requests are checked first. Once a nearby fire is detected, the request handler activates to manage the fire-extinguishing mission. If no immediate fire is found, the NPC engages in exploratory actions. Upon discovering a fire, the `MOVETo(fire)` guides the NPC to approach it, and then the `DOUSE(fire)` node initiates the extinguishing process. Should additional assistance be required, the HRS node sends out a hard request to squad members listed, waiting for two acknowledgments within a 5000-millisecond timeout. Upon receiving confirmations, the `DOUSE(blackboard)` performs the extinguishing action on the specific fire referenced by the ”nearbyFire” key on the blackboard. The setup of this scenario is designed to simulate individual exploration behavior and the coordinated behavior required for extinguishing fires.

DOI: 10.54254/2755-2721/2025.30674

In the game created by the authors, they demonstrate the difference between using hard coding or using coordination nodes to enable NPCs to complete collaborative tasks. However, it is regrettable that they did not conduct a complete set of comparative experiments and record data.

4. Emotional Behavior Trees

According to [2], emotion factors are an extension to behavior trees, in an attempt to enhance their unpredictability and the degree to which they resemble humans. [2] proposes an advanced priority selector which result dynamically changes based on the current state of emotion variables, thus simulating emotions and its influences in a behavior tree's decision-making progress.

According to [3], emotions affect the way in which humans value the outcome of their actions, as they will tend to select choices that maximizes positive emotions (i.e. happieness) and minimizes negative emotions (i.e. anger and fear), potentially causing optimal choices to be neglected, which is nonetheless a process of human decision making.

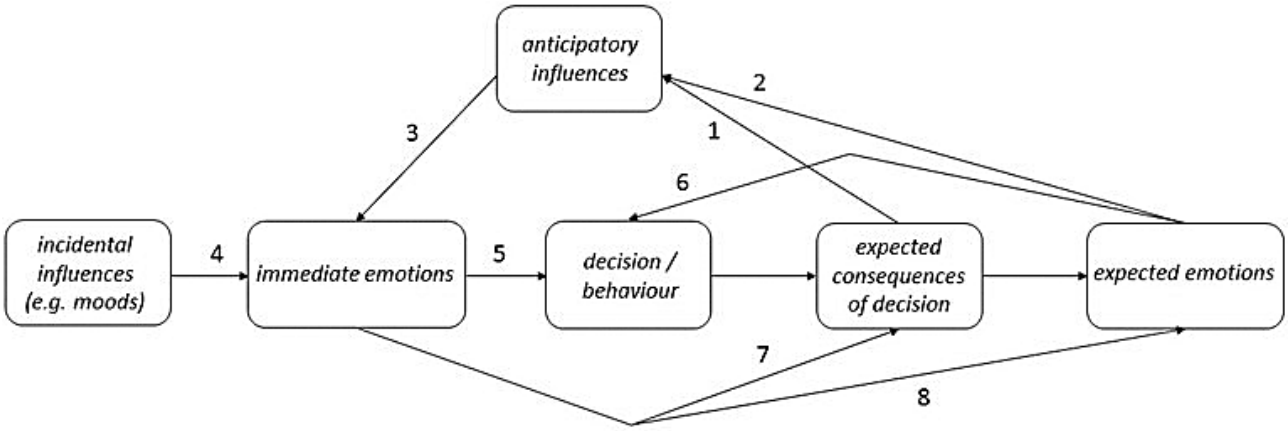


Figure 3: Loewenstein's propsed schema of emotion and decision making.[3]

In this model, upon making an decision, an individual's consideration of the immediate expected consequences creates anticipatory influence that initially affects the person's emotions, which, in combination with incidental influence from external sources, influences the final decision made, which furthermore affects the individual's emotions post-decision. This self-influencing flowchart-like structure provides an entry point and a foundation to Johansson's[2] implementation of simulated emotions in behavior. Apart from the central process of decision making, emotions also exerts influence in other commonly seen tasks, including - risk perception, which is largely related to the emotion of fear; time discount, the amount of time that must be spent on the completion of a task, is tied to the emotions an individual is experience at a given time as well as that individual's characteristics, moods, and emotional values, introducing more elements of uncertainty and vagueness to the process; mental strains and fatigue, which negatively affects an individual's ability in tasks such as planning that requires a broader picture of the situation.

Johansson proposes the Emotional Selector, an extension to the regular selector node that determines execution priority of its child nodes by taking the aforementioned variables of emotion (decision making, risk assessment, time considerations, mental and emotional issues) into consideration.

DOI: 10.54254/2755-2721/2025.30674

4.1. Planning Effort

Most planning processes involving assessing each action in the planned path in a linear fashion, thus possessing no information on the actions that are yet to be considered. In contrast, all possible actions and their corresponding conditions are specified initially be a behavior tree, and therefore providing no space for planning. Therefore, in order to simulate planning behavior and assess the effort required to plan for an action, the concept of planning value, a concrete value bound to each individual action, is introduced. This value is to be set by the developer of the behavior tree, and in general, more complex tasks would require more planning, and would thus receive a higher planning value.

Planning in a behavior tree that incorporates emotional selectors involves computing the sum of planning value of each node in the proposed path of selection. Nodes receive planning values as follows:

1) Actions: Planning value set by the developer.

2) Conditions: Conditions receive a planning value of 0 as they involves simple true-or-false situations, and if a condition is to be evaluated to false, the corresponding path would no longer be considered a viable path. Therefore, conditional nodes do not affect the planning effort and are excluded from the planning process.

3) Sequence Nodes and Parallel Nodes: Both type of node's planning value is equal to the sum of the planning value of all of its children as it executes all its child nodes.

4) Priority Nodes: Priority nodes: A priority node's choice of child node to execute is never known due to its selecting logic being only accessible by itself and its implementation, therefore its planning value is assumed to be the average of all of its child nodes' planning value.

4.2. Risk Assessment

Risk assessment involves assigning a risk value indicative of the current situation the bearer of the behavior tree is in, which may be affected by their emotions at a given point in time. The risk value, being a floating-point value in the interval of [0, 1] functions similarly to the planning value, represents the probability of risk at a given point in time. 0 represents no risk is associated with the desired action, while 1 represents significant risk or danger.

1) Actions: The risk of an action is to be assigned by the developer.

2) Conditions: In general, conditions carries no risk (with a risk value of 0).

3) Sequence Nodes and Parallel Nodes: The risk value of such nodes is the sum of all of its children nodes' risk value.

4) Priority Nodes: Since the child node that such a node will execute remains indeterminate, a priority node's risk value is the average of all of its children nodes' risk values.

4.3. Time Effort

When making decisions, individuals lean towards options that is the least time-consuming, however, this preference may be affected by an individual's emotions at a given point in time, as well as that individual's characteristics and emotional tendencies. The time required to execute a node is represented by an interval $[L, U]$, denoting the minimum and maximum time.

1) Actions: The time to complete an action is set by the developer.

2) Conditions: Conditions are instantaneous and always possesses a time effort value of $[0, 0]$.

3) Sequence Nodes: The lower and upper limits are the sum of each of its child nodes' lower and upper time effort values, respectively.

DOI: 10.54254/2755-2721/2025.30674

4) Priority Nodes and Parallel: Priority selectors only execute one of its child nodes, and parallel nodes executes all of its nodes in parallel, thus the lower and upper limits of such nodes are modeled by $[L, U] = [min\{L_1...L_n\}, max\{U_1...U_n\}]$.

4.4. Emotions

With the methods of calculating planning, risk, and time established, emotion weights are now introduced to further the variability of the decision making progress. The authors define emotions that positively or negatively affects the aforementioned aspects of decision making. For instance, emotional weight for *risk*, with positive emotions $r^+_{1...n}$ and negative emotions $r^-_{1...m}$, can be modeled as the average of all negative emotions subtracted from the average of all positive emotions.

Weights for planning effort and time effort can be modelled similarly. For each child node of the emotional behavior tree, we then calculate its weights based on the formula proposed above. A formula that simulates time discounting in humans, a behavior that causes individual to evaluating lower values in scenarios that involves a delayed reward, is also incorporated into the evaluation of time effort. In the end, the emotional selector performs the action that has the lowest weight, which represents the least amount of resistance. A probability factor is also added, which allows for further variability and allows for errors to occur by selecting non-optimal nodes.

4.5. Example

Johansson proposes an example that uses the proposed emotion behavior trees. Consider the following behavior tree:

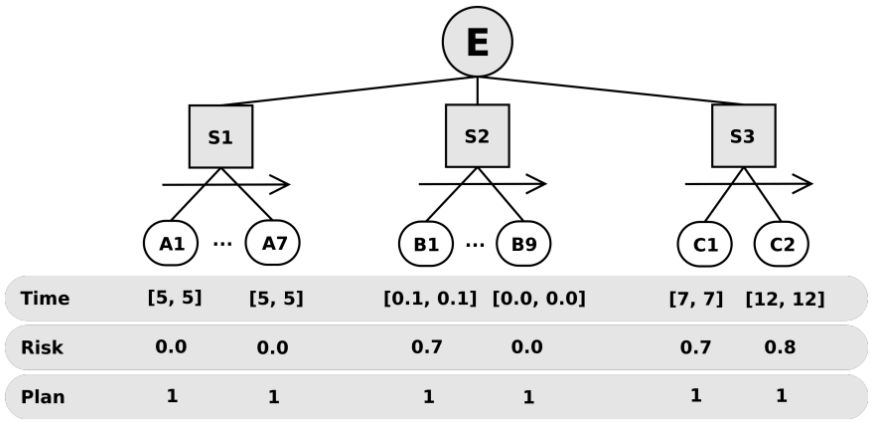


Figure 4: An example of an emotional selector node.[2]

The authors introduce three emotions that affects the final weight of nodes: *fear*, a negative emotion, *sadness* and *fatigue*, both positive emotions. In each of the below cases, the value of one of the three emotions is set to 1.0 while all of the remaining are set to 0.0. The following weight values would be produced:

DOI: 10.54254/2755-2721/2025.30674

Table 1: Fear

	W*risk*	W*time*	W*plan*	W
S1	0.0	0.193	0.162	**0.355**
S2	0.7	0.0148	0.169	0.884
S3	0.94	0.188	0.109	1.240

Table 2: Sadness

	W*risk*	W*time*	W*plan*	W
S1	0.0	0.966	0.162	1.130
S2	0.0	0.0741	0.169	**0.243**
S3	0.0	0.938	0.109	1.050

Table 3: Fatigue

	W*risk*	W*time*	W*plan*	W
S1	0.0	0.193	0.808	1.000
S2	0.0	0.0148	0.844	0.859
S3	0.0	0.188	0.545	**0.733**

The above tables shows the calculated emotional weights in each of the decision making progress for the emotions *fear*, *sadness*, and *fatigue* when that emotion is amplified. The highlighted cell denotes the final option selected.[2]

A correlation between the amplified emotion value and the outcome can be seen (for instance, enhanced fear leads to an individual choosing the least risky option above all other options), thus verifying the simulation.

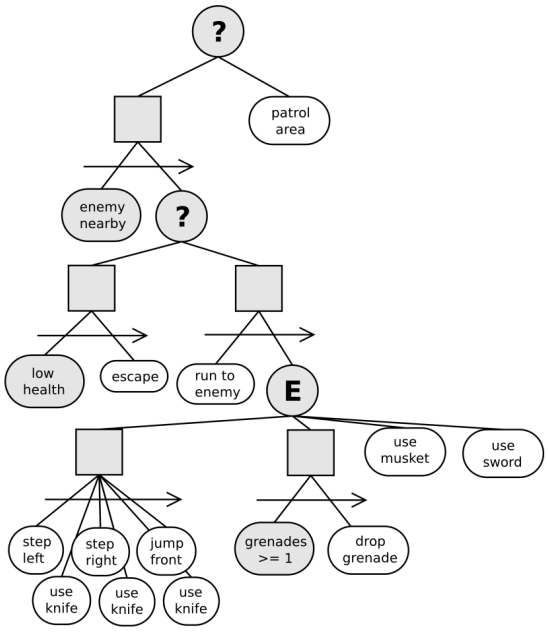
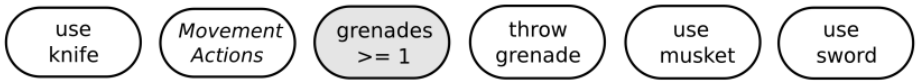


Figure 5: A behavior tree that contains an emotional selector node, indicated by the node marked **E**.[2]

DOI: 10.54254/2755-2721/2025.30674

4.6. Case Study - Fighting NPC

Consider the behavior tree for a NPC in a computer game as shown in Fig. 5. The emotional selector in this example determines the method with which the NPC will attack its targets. Each action's time, risk, and planning value are shown as follows:



	use knife	Movement Actions	grenades >= 1	throw grenade	use musket	use sword
Time	**[0, 0]**	**[0.33, 1.0]**	**[0, 0]**	**[1.5, 2.5]**	**[7, 10]**	**[2, 3]**
Risk	**0.033**	**0.0**	**0.0**	**0.7**	**0.2**	**0.6**
Plan	**1**	**1**	**0**	**1**	**1.7**	**1**

Figure 6: Actions that may be selected by the emotional selector node, along with their respective planning, risk, and time values.[2]

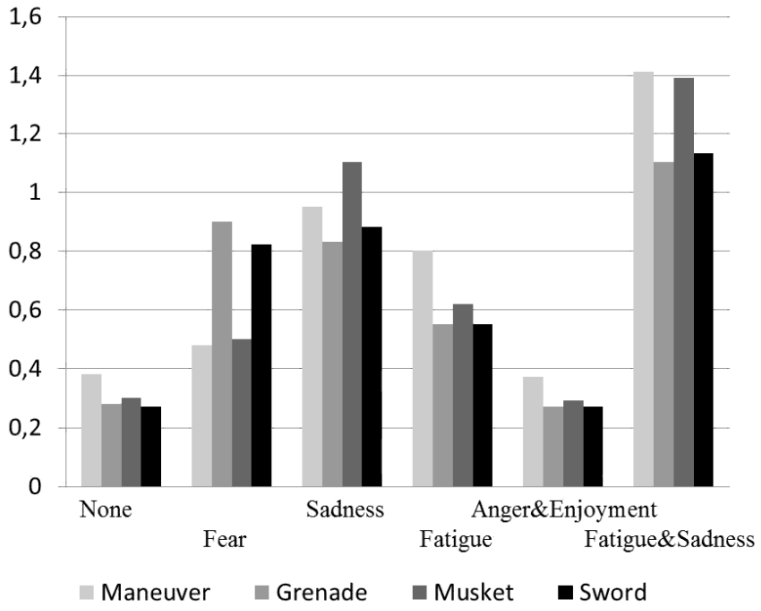


Figure 7: Final weight on each action under different sets of significant emotions. "Maneuver" represents the sequence selector note depicted in Fig. 5 which executes *Knife* and *Movement Action* actions in order. A lower value indicates a higher likelihood of the emotional selector node selecting that option.[2]

1)Knife: A swift attack, using little time and involves little risk. Performed by a sequence selector node, this action has a combined risk value of 0.1, a total planning value of 6 and a time interval of $[0.99, 3]$.

DOI: 10.54254/2755-2721/2025.30674

2)Grenade: Throwing an explosive represents a high risk as the explosive poses a threat to the user himself.

3)Musket: Muskets, despite being usable from afar, takes a long time to load and prepare, thus having a high time interval and planning amount.

4)Sword: Planning value and time interval are small, but poses a relative large risk of 0.6. In this scenario, we introduce the emotions fear, fatigue, sadness, anger, and enjoyment. The following data is collected from this test:

It is evident that the decisions made in this scenario is indicative of the NPC's current emotion values. For instance, when fear level is high, the result of the selector leans towards options involving less risk, while faster actions that requires less planning are preferred when the NPC experiences fatigue.

4.7. Conclusion

Emotional behavior trees brings an innovative approach to extending the simple behavior trees. By introducing emotional factors and weights to affect core processes of decision making, a new degree of variability and human-like responsiveness can be enabled in behavior trees. Yet this model only attempts to simulate human emotions over a select set of behaviors, namely planning effort, risk, and time effort. Such a generalization is most likely not all-encompassing of human emotions, which by nature is vague, abstract, and yet to be fully understood by the science community and humanity. Such an attempt proposed at simulating emotions in automated behaviors is sufficient to serve as a solid foundation for further explorations and improvements to behavior trees.

5. EvolvingBehavior: Towards Co-Creative Evolution of Behavior Trees for Game NPCs

The game developer always wants their NPCs more adaptable in the game environment in order to provide immersive experience for players. So the NPCs are created carefully focusing on creating a specific player experience. Actually, players sometimes interact in unexpected ways with games, and may want help developing NPCs which could handle unusual gameplay situations.Tools that could generate behavior trees might support designers in game development.

The paper [4] presents EvolvingBehavior, a novel tool that employs genetic programming to evolve modifications to hand-crafted behavior trees. As an initial evaluation of the tool, the paper describes an experiment that compares evolved behavior trees to trees hand-designed by their researchers and to randomly-grown trees for zombies in a 3D survival game. After that the paper explores the results through quantitative evaluation and in-depth discussion. They find that the evolved trees exhibit clearly stronger similarities to hand-designed trees than randomly-grown trees [4].

5.1. Background about Genetic Algorithm

Genetic algorithm is a series of search algorithms inspired by the theory of natural evolution. By mimicking the processes of natural selection and reproduction, genetic algorithms can provide high-quality solutions to a variety of problems involving optimization and learning. At the same time, they resemble natural evolution and can therefore overcome some of the obstacles encountered by traditional optimization algorithms.

Individuals in a population contain some of the elements required for an optimal solution. Selection and crossover processes convey these elements of an individual to the next generation, while possibly combining them with the essential elements of other optimal solutions. This creates genetic

DOI: 10.54254/2755-2721/2025.30674

pressure that leads more and more individuals in the population to include elements that constitute the best solution.

5.2. Customized Genetic Algorithm in This Paper

5.2.1. Approach of Evolution

To simulate the evolution process, the following key indicators are needed to be determined to ensure that the evolution process is reasonable, useful and efficient.

Representation

The paper translate the built-in behavior trees in Unreal Engine into a compact tree structure, a “chromosome” [4]. This chromosome has the ability to store references to specific, predefined nodes (called ”mapping nodes”) or templates that generate nodes with different attribute values (called ”generating nodes”). This representation is reasonable because every chromosome is a compact tree structure to easily simulate combination of genes. Pursuing perfect linear structure is not required due to the fact that chromosome is the carrier of mutation.

Flexibly Measure Fitness

Adaptive assessments should be designed to determine what kind of individuals are needed in the evolution. When doing this process, the emphasis is on flexibility, because different games refer to different important points and ”environmental” conditions. Measuring Fitness is perhaps the most critical part of the system, the aspect that determines whether the results are helpful. Thus, no single fitness measure will suit every game, and each game must define its own.

Selecting Parents

How to choose parents is a strategy that affects both environmental adaptability and anti-risk capability of a population. If only individuals with high fitness are selected as parents, the population will be highly adaptable at this time. However, it may be at risk of extinction when the environment changes due to the lack of diversity. Thus, the population doesn’t equip with anti-risk capability. And vice versa, so it’s critical to find a balance in EvolvingBehavior to evolve the behavior tree.

The paper adopted a tournament selection method with a tournament size of 4, which determines the intensity of selecting individuals with high fitness and also affects the diversity of the next generation. The most fit individual is then selected as the parent. This method is simple, adjustable and adaptable, which is suitable for behavior tree evolution system and allows designers to control system flexibly [4]. The system combines parental selection and population renewal, replacing the entire population with each generation. Through elitism, several individuals with the most potential from the previous generation are retained and copied directly to the next generation to avoid losing information on the most successful individuals due to random variation and to reduce population swelling.

Reproduction and Mutation

After selecting two parents, new offspring are generated by copying the parents. The offspring may undergo crossover and/or point mutation. The system allows multiple types of mutations for each offspring, each with its independent probability, giving designers greater control over the mutation rate.

5.2.2. Crossover

The basic form involves selecting any random subtree from the child and replacing it with a random subtree from the second parent, simulating genetic recombination. There is a bias toward selecting internal nodes as crossover points to increase the average amount of tree exchanged, affecting behavior. The crossover operator selects nodes from specific tree depths, with the depth chosen uniformly

DOI: 10.54254/2755-2721/2025.30674

at random. It is difficult to find a corresponding point on both individuals, so the subtrees to be related in position, or even size are not required [4].

5.2.3. Point Mutation

Several types of point mutations are implemented: replacing, adding, or deleting random nodes. Traditionally, the entire tree is traversed, and each node has a small probability of mutation. In the current system, random point mutations are applied to a single node. Even if the internal node is deleted, it is normal in the evolution process like deformed individual in biology.

5.3. Evaluation of Evolved Behavior Tree

5.3.1. Evolution Effect Experiment

In the sample game for test, the player must survive for as long as they can in a level populated with simple zombie-like enemies. The enemies, in the standard game, can patrol a preset route, notice the player through sound or vision, and chase the player [4].

According to the principle of Flexibly Measure Fitness, a segmented weighted linear fitness function was designed to reward zombies for moving and patrolling the map, chasing humans and causing damage, and searching the vicinity of the last known target location after losing the target. A penalty mechanism was also designed to penalize zombies that did not move and those with behavior trees that were too complex or too simple, reducing the likelihood of bloat and simple trees [4].

As presented in the Fig. 8, researchers established weighted settings for the fitness scores of each zombie. This approach was used solely for experimental purposes within the game, and actual development requires custom design and adjustment. In the weights, TotalChaseTime was assigned a weight of 100, while HasNeverMoved received a weight of -500, highlighting the encouragement for NPCs to patrol and pursue players more frequently.

There are different map sizes, number of human characters and number of zombies. And a total of nine experimental conditions were evaluated over 1000 generations.

Quantitative fitness comparison and qualitative evaluation are adopted in the paper. For quantitative comparison, the researchers adopt fitness score to compare the fitness for the trees in each experimental condition, comparing the manual trees, random baselines and evolved trees [4]. The fitness score represents the performance of the behavior tree under experimental conditions, and the higher the score, the more consistent the behavior tree is with the design goal. The qualitative evaluation part is to observe the performances of three type of behavior trees in the actual game scene and analyze its behavior logic.

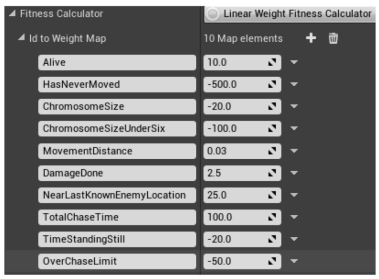


Figure 8: The identifiers and weight settings for the piece- wise linear fitness function used in researchers' experiments [4].

DOI: 10.54254/2755-2721/2025.30674

5.3.2. Comparison and Evaluation

Quantitative Fitness Comparison: Fig. 9 and Fig. 10 show that the fitness scores of the evolved behavioral trees are significantly higher than those of the random baseline trees in the Medium Map and Large Map conditions. In particular, in Fig. 9, the fitness of the evolutionary tree gradually rises with the number of generations and reaches stability around 1000 generations, showing better tracking and patrolling behavior than the random baseline tree. This is further supported by the Fig. 10, which show that the median fitness of the evolutionary tree is much higher than that of the random baseline tree, and the distribution is more concentrated, indicating a more stable overall performance.

However, there was little difference in fitness between the evolutionary tree and the random baseline tree in the Small Map condition, suggesting that evolution did not significantly improve the performance of the behavior tree in these specific scenarios. This may be related to the complexity of experimental conditions and the design of fitness function [4].

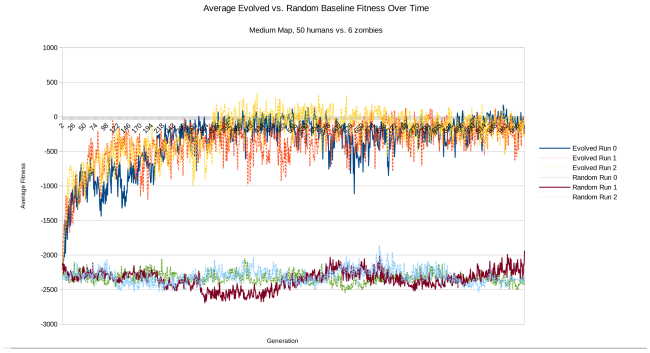


Figure 9: Graph of the average fitness per generation of the evolved vs. the random baseline zombies, on the Medium Map, with 50 zombies and 6 humans [4].

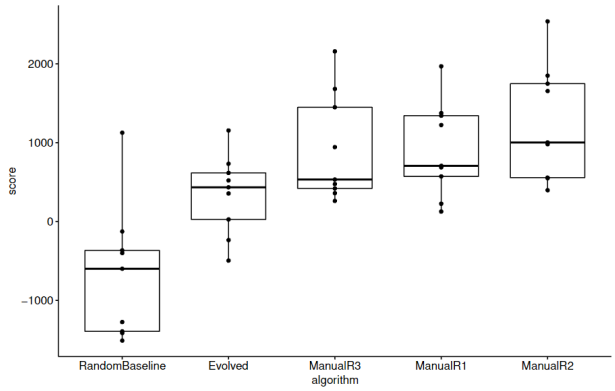


Figure 10: Box plots of the average fitness results for each algorithm across all problem configurations [4].

Qualitative Evaluation: The observation results show that evolutionary trees exhibit intelligent behavior similar to hand-designed trees in many cases, such as effectively switching between

DOI: 10.54254/2755-2721/2025.30674

patrolling and tracking, and conducting reasonable searches after a lost target [4]. These behaviors suggest that evolutionary trees can not only achieve the goals set by experiments, but also exhibit rational strategies of action in complex environments.

5.4. Summary

Behavior trees generated through natural evolution can better simulate, and even surpass, manually designed behavior trees, enhancing development efficiency and saving manpower. However, it is important to first determine the details of the fitness function, which should be designed based on the focus of each game. An appropriate fitness function can yield significantly better results. Secondly, the strategy for selecting parents is crucial. This experiment used a tournament selection strategy, but in actual development, various strategies such as roulette wheel selection, tournament selection, and rank selection can be tried to find the optimal solution. In summary, it is essential to analyze the NPC behavior standards according to the specific game. For the tool developed by this experiment, the research shows that, in appropriate conditions, the tool was capable of evolving NPC behavior that was noticeably similar to hand-designed behavior, though with some differences and variability [4].

6. Conclusion

This paper examines innovative extensions to behavior trees in computer games, enhancing NPC responsiveness and human-like interactions. We explored three key advancements: Event-Driven Behavior Trees (EDBTs) for multi-agent coordination, emotional behavior trees for simulating human emotions, and genetically evolved behavior trees to improve adaptability. EDBTs enable sophisticated NPC interactions through coordination nodes, improving scalability and reusability. Emotional behavior trees integrate simulated emotions, allowing NPCs to exhibit human-like decision-making processes. This introduces a new dimension of variability and realism in gaming experiences. Lastly, genetically evolving behavior trees through adaptive mechanisms allow NPCs to evolve naturally, offering complex and authentic behaviors. These advancements significantly enhance the immersion and realism of NPCs in games, contributing to a more engaging player experience. Future research can build on these foundations to further refine NPC behavior, driving innovation in game AI development.

Acknowledgment

Siyun Guo, and Sitong Hou contributed equally to this work and should be considered co-first authors.

References

[1] Ramiro A Agis, Sebastian Gottifredi, and Alejandro J García. “An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games”. In: *Expert Systems with Applications* 155 (2020), p. 113457.

[2] Anja Johansson and Pierangelo Dell’Acqua. “Emotional behavior trees.” In: *2012 IEEE Conference on Computational Intelligence and Games (CIG)* (2012).

[3] George Loewenstein and Jennifer S. Lerner. “The role of affect in decision making.” In: *Handbook of affective science 619.642* (2003).

[4] Nathan Partlan et al. “EvolvingBehavior: towards co-creative evolution of behavior trees for game NPCs”. In: *Proceedings of the 17th International Conference on the Foundations of Digital Games*. 2022, pp. 1–13.